Docket Number: POU920010061US1

Inventor: Dennis A. Quan, Jr.

Title: METHOD AND PROGRAM PRODUCT
FOR STRUCTURED COMMENT ASSISTS
IN COMPUTER PROGRAMMING

APPLICATION FOR UNITED STATES

LETTERS PATENT

INTERNATIONAL BUSINESS MACHINES CORPORATION

METHOD AND PROGRAM PRODUCT FOR STRUCTURED COMMENT ASSISTS IN COMPUTER PROGRAMMING

Field of the Invention

[0001]   This invention relates to the field of computer programming tools and in particular to an intelligent real time tool to assist a programmer during the writing, evaluation, and/or maintenance of a computer program.

Background of the Invention

[0002]   Two related problems exist in the field of computer programming that include, but are not limited to, generating a computer program quickly and accurately on a first attempt, and maintaining a computer program with a minimal amount of effort once a computer program exists. Program listings are created and maintained by executers which are themselves programs. These executers (including, but not limited to, program editors, interpretive source code execution applications, source code compilors and simulators for example) are programs that operate on program listings (usually by interpreting the program statements in the program listing) and normally provide a Graphical User Interface (GUI) to a human programmer. Other kinds of executers exist including programs that interpretively execute the statements in a program listing, The two related problems exist due to many factors that include, but are not limited to, the increasing complexity of computer programs generally, the architectural modularity of computer programs, and the increasing distribution of programmers that are contributing to a common program across campuses, countries, and even continents. Each of these factors places a premium on the efficiency of a programmer whose role is to develop and/or maintain a computer program. Various methods are known in the art to address the problem. For example, US Patent 6,026,233 "Method and apparatus for presenting and selecting options to modify a programming language statement" (Schulman et al.), which is incorporated herein by reference provides a method for a program editor to assist a programmer so he doesn't have to rely on extraordinary means to correctly code a program.

[0003] The referenced Schulman patent generates automatically and/or manually invoked assist windows that contain information applicable to a programming language statement that is proximate to the present location of the character position cursor. The assist window information can be used to complete at least one portion of a programming language statement being constructed by the programmer. The assist information can also be used by the programmer to obtain help that is relevant to the immediate portion of the programming statement by supplying information relevant to the present location of the character position cursor in the immediate programming language statement.

[0004] The prior art doesn't provide for the case where a first version of a program architecture doesn't provide a function that could enhance the functionality of the language. When the shortcoming is discovered, it may be overcome by adding architected function to the program architecture in a later release of the program. The problem exists that in this case, the needed function isn't provided in the earlier released environment and is only provided in unique code statement enhancements to the program for the newer environment, thus the new function code is not useable in both the new environment and the old environment.

[0005] An example of the problem is found in an early version of ECMAScript which is a "type"-less language. A programmer using ECMAScript must separately keep track of statement variable type. A later version of the code provided type-ing statements but the use of these rendered the code unusable in environments that used the old level of executer. A method is needed in ECMAScript to provide type-ing in a way that it doesn't interfere with execution in an old level of executer and still provides type-ing in the new levels of executers.

[0006] Object Oriented "type" manipulation is described in US Patent 6,202,202 (Steensgaard) "Pointer analysis by type inference for programs with structured memory objects and potentially inconsistent memory object accesses" assigned to Microsoft Corp. US 6,202,202 is hereby incorporated by reference. The Steensgaard patent discloses a pointer analysis by type inference for a computer program with structured memory objects and potentially inconsistent memory object accesses helps approximate run-time store usage for the program. The analysis represents

locations for the program with types describing access patterns for the represented locations based on how the locations are accessed in the program. The analysis describes access patterns for structured memory objects, elementsof structured memory objects, and memory objects accessed in inconsistent manners in the program. The analysis identifies store usages described by the program and determines whether the location(s) and/or function(s) affected by the identified store usages are well-typed under typing constraints. If the identified store usagesare not well-typed, the analysis modifies types for location(s) and/or function(s) affected by the identified store usages as necessary so the store usages are well-typed. When the locations and/or functions for all identified store usages are well-typed, the program is well-typed with the set of types defining a store model for the program.

Exemplary Computing Environment--(FIGURE 1)

[0007]   Figure 1 illustrates a block diagram example of a general purpose computer system 100 that is suitable for use with the executer. However, the executer is operable in any of the several computing environments that can include a variety of hardware, operating systems, and program modules that are all commercially available in the industry. Program modules include, but are not limited to, routines, programs, components, data structures, and the like that perform particular tasks and/or implement particular abstract data types. Moreover, persons skilled in the art appreciate that the executer can be practiced with other computer system configurations including, but not limited to, hand-held devices, network computers, multiprocessor based systems, microprocessor-based or other general purpose or proprietary programmable consumer electronics, minicomputers, mainframes, and the like. The executer may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through communications networks. In a distributed computing environment, program modules may be located in and/or executed from local and/or remote memory storage devices.

[0008]   The executer and any other necessary programmed instructions and/or commands are executable on processor 102. Processor 102 stores and/or retrieves programmed instructions

-3-

and/or data from memory devices that can include, but are not limited to, Random Access Memory (RAM) 110 and Read Only Memory (ROM) 108 by way of memory bus 152. Another accessible memory device includes non-volatile memory device 112 by way of local bus 150. User input to computer system 100 is entered by way of keyboard 104 and/or pointing device 106. Human readable output from computer system 100 can be viewed on display 114 or in printed form on local printer 115. Alternatively, computer system 100 is accessible by remote users for purposes that can include debugging, input, output and/or generating human readable displays in printed and/or display screen output form, or any other output form, by way of a Local Area Network (LAN) or Wide Area Network (WAN) 116.

[0009]   Certain attributes of a program language may resist the previous information tools since the program language may not incorporate the assist functions required. Program Editors for some languages such as C++, Visual Basic, and Java take advantage of variable type declarations to determine the "types" of variables defined by the programmer. Examples are:

Dim X as Form (Visual Basic) defines "X" as having the type "Form"

Ostream* os; (C++) defines "os" as having the type "Ostream"

Java.io.Reader reader; (Java) defines "reader" as having the type
"Java.io.Reader"

[0010]   By knowing the types of variables, the editor can determine the correct choices to be presented in the assist window. However, in languages without variable type declarations (such as ECMAScript), it is impossible for the editor executer to determine the types of programmer defined variables. Unlike statically-typed languages, where types can be inferred from code statically, ECMAScript uses dynamic typing (i.e. Runtime type determination), which makes it computationally not feasible to determine these types during editing.

-4-

[0011]   Often it is the case, however, that ECMAScript is used in conjunction with certain objects made available by the "host" of the ECMAScript interpreter whose types are fixed. An example is the use of ECMAScript in a web browser. Objects such as window and document are exposed to ECMAScript within an HTML document from the host of the ECMAScript executer, namely, the web browser. These objects have fixed type as defined by the W3C HTML DOM standard.

[0012]   Java language uses special comments to provide a way to generate source code documentation (Javadoc).   As exemplified by the following:

/** This is a javadoc comment @author Dennis Quan */

[0013]   Java uses a double asterisk at the beginning of a comment field to identify a special comment that will be used by Javadoc to generate documentation from a source code listing. This allows a tool (separate from the Java compiler) called javadoc to parse the source files and automatically generate source code documentation, a useful thing to do since programmers aren't likely to document their code in separate documents. Notice the use of the @author keyword to designate which part of the comment includes author information. This is a use of comments for creating human-readable documentation. It doesn't provide any functionality to the program listing.

[0014]   Special comments are used by Microsoft Java/Windows/ActiveX integration to extend their version of the Java language to support ActiveX (ActiveX is Microsoft's proprietary component technology). Since the Java compiler is already aware of (but does little with) Javadoc comments, they have used this as a convenient basis for their proprietary language extensions. In this sense the /** .. */ comments are similar to preprocessor instructions in C or C++ (e.g., #include, #define, etc.) In that they are directives meant to be processed **before** the compiler sees the code.

[0015]   An example of the ActiveX format is:

-5-

```
/** @dll.import("KERNEL32") */

public static native int GetEnvironmentStrings();
```

[0016] These special commands, instead of being used for documenting a function, are used to declare to the compiler some attribute of the function (for example, that the function GetEnvironmentStrings is implemented in a library named KERNEL32). These comments are mandatory. They provide functionality for the context of the program. The function within these special commands must be interpreted for the program to function correctly. As such they could be considered an addition to the syntax of the language. Here are the key differences.

[0017] ActiveX special commands fail to provide means to specify type declarations. Instead, ActiveX provides a means for specifying where the function is implemented (in a native Windows-specific library usually). In ActiveX, an incorrect special command specification could cause compiler errors or cause the program not to execute correctly; hence, it is modifying the semantics of the language. ActiveX doesn't provide means for backward compatibility in that the special commands are required for proper execution of the code, they haven't addressed the need to provide code extensions that permit new functions to be performed in new execution environments while keeping functionality in old execution environments. Therefore, ActiveX function could have been equally performed using architectural modifications (new Java syntax).

[0018] Microsoft recognized the deficiency of ECMAScript language (in that it didn't provide type declaration capability). They solved this in JScript.NET (their new implementation of ECMAScript) by introducing new syntax in order to solve the problem. Their new syntax however does not work with earlier versions of ECMAScript executers and is nonstandard.

[0019] An editor is thus able to give syntax assistance in the specific circumstances where one of these predefined host objects is used since the host object has a predefined type. It is not, however, able to give assistance when programmer defined variables are used, even when it is completely obvious. For example:

var x = window;

[0020] By declaring x to refer to the window object, it is evident that x now has the same type as window. However, the editor cannot make this deduction in general because of the dynamic typing of ECMAScript. In addition, there is no way in this language (as in other scripting languages) to inform the editor of the type, as can be done in Visual Basic using the Dim keyword for example.

[0021] For example, in a program language, "Standard" ECMAScript program language had no facility for specifying a variable type to a variable:

var x = expression_a;

[0022] In the above example, expression_a represents a value that has no predefined type. A more recent release incorporates a method for a programmer to assign a variable type, thus:

var x : Window = window;

[0023] This solution is incompatible with the previous level of ECMAScript where such an expression would not be supported for assigning type. A method is needed that would allow programmers to interact with existing program structures with enhanced tools.

Summary of the Invention

[0024] The foregoing problems and shortcomings of the prior art are addressed and overcome and further advantageous features are provided by the present invention.

[0025] In certain environments, program code (listing) instructions are deficient. An example is in object oriented programming when parameters in the object to be coded are defined by other program objects. In this case, the programmer must make sure that his code is consistent with

other modules. Another example is that it is desirable to be able to use new versions of executers (editors, simulators and program interpretive execution applications for example) on old versions of program listings, while taking advantage of functionality of a newer version. Another example is that it is desirable to add functionality to a program listing for special environments without disturbing the basic function of the program. In these cases, if the program were modified to perform the functions, it would no longer be compatible with it's intended environment.

[0026]   The present invention utilizes special program listing statements that are ordinarily ignored by executers (compilers, simulators and interpretive program execution environments for example). In most program languages, the comment field can be used to provide the special program listing statement, but the invention is not restricted to the use of comment field. The invention could also be practiced by incorporating a table space that is not referenced directly by the program. The table space would relate special comment function to the appropriate location in the program listing. Furthermore, the table space could be implemented in a file separate from the program listing altogether. The comment field embodiment is used to provide an example of how one would implement the present invention.

[0027]   Accordingly, the comment field of a programming architecture is transformed into an operational statement (or statements) by the present invention.

[0028]   While the present invention could be practiced by transforming all comments in a program listing to executable statements, in the preferred embodiment, it is advantageous to provide conventional comments in combination with the executable comments (herein referred to as "special comments"). Thus, the present invention further defines a subset of the comments as executable. This is preferably accomplished by structuring the special comment in a way that uniquely differentiates it with conventional, non-executable comments. In a preferred embodiment the special comment field contains predefined special characters in a predefined location within the conventional comment format. Thus, in this embodiment, an ordinary comment contains special characters to identify it as a special comment and further contains

information preferably such as instructions, data, instruction modifiers and the like according to the needs of a programming environment.

[0029]   The present invention permits a programmer to embed special comments of a specific form into a programming language. The special comments are then recognized, by a compiler for instance, to provide special functionality that is not ordinarily supported by the program language.

[0030]   An example that is shown for explanation is as follows:

In the current release of ECMAScript, static type declaration is not supported. The invention provides for a special comment field to imbed type declarations in the code listing in comment fields. Tools are then provided to support the special comment type assignment in Type Managers, Expression Parsers, Comment Parsers, and Comment Auto-inserters as if they were part of the native program language. The special comments incorporate a special format to differentiate them form ordinary comments. For example, in many languages a comment is defined by /* followed by commenting text ended by */. Thus the expression:

```
/*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*/
```

Is a comment incorporating a string of x's. In the present invention, a predefined convention is used that reserves certain comments as the special comments. An example might be a comment field beginning with "$$". Thus;

```
/*$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*/
```

Becomes a special comment.

-9-

[0031] Special comments of this invention can generally be thought of as extensions to program listing executables. These special comments are detected and executed by special compilers that are designed to execute special comment function that is not incorporated in the native programing architecture. Such special compilers may be used in program editors or source code executers for example.

[0032] It is therefore, an object of the present invention to provide special function to a programming language comment field using a unique comment field.

[0033] It is another object of the present invention to provide special function facility to assist a program editor.

[0034] It is another object of the present invention to provide interpretivly operational content to a programming language comment field.

[0035] It is a further object of the present invention to use special function comments to define type to a variable.

[0036] It is yet another object of the present invention to provide a method to assist a programmer in creating special function comments.

[0037] It is still another object of the present invention to provide a method to automatically apply special programming language comments to a program listing.

Brief Description of the Drawings

[0038] The subject matter which is regarded as constituting the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features and advantages of the invention are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

FIGURE 1 illustrates a block diagram of an exemplary computing environment in which the present invention can be implemented;

FIGURE 2 is a flow chart that demonstrates components of a preferred program editor;

FIGURE 3 is a flow chart that demonstrates components of the present invention in a preferred program editor;

FIGURE 4 is a flow chart that demonstrates an embodiment of the present invention in an interpretive execution or simulation environment;

FIGURE 5 is a flow chart that demonstrates an embodiment of the present invention using two versions of an executer;

FIGURE 6 demonstrates a prompt assistance sub-window according to the invention;

FIGURE 7, and 8 demonstrates providing a special comment to a program listing;

FIGURE 9 is a flow chart demonstrating creating a special comment in an editor; and

FIGURE 10 is a flow chart demonstrating prompt assistance for creating special commands.

Detailed Description of the Preferred Embodiment

[0039] Referring to Figure 2 a preferred embodiment of the present invention is depicted in a program editor environment. A conventional program listing **2101** includes program instruction statements A, B, C, D and E. The listing also includes program comment statement X. The program listing is displayed on terminal **2002** as display **1700**. Operations **2108** are performed on the program listing by the editor executer **2100**. These operations include, but are not limited

-11-

to Manual Edit **2109**, permits the programmer to directly enter statements that are incorporated in the program listing **2101**; AutoInsert **2110**, incorporates statements in the program listing **2101** automatically according to a predetermined functionality of the editor; and, Attribute Prompt **2111**, provides display prompts (ordinarily sub-windows) to assist the programmer in performing manual editing **2109**.

[0040] The editor conventionally parses **2102** the program listing statements **2101** and provides the parsed program to a compiler **2103** that interacts with the editing function **2108** to assist the programmer in editing his program. The compiler uses an attribute record **2104** to keep track of the current state of the statements in the program and a type manager **2112** to provide type information to the compiler.

[0041] In Figure 3, an embodiment of the present invention, the program listing **2102** parser in the executer **2200** includes a comment parser **2201b** as well as a conventional parser **2201a** . The comment parser ignores conventional comments and sends statements found in special comments to the compiler **2103** and the editor components **2108**. In this embodiment, the attribute record **2104** keeps records of attributes according to the program block that they are related to. Parent attribute record **2105** contains global attributes and child1 attribute record **2106** contains attributes associated with a block of the program delineated by brackets "{" and "}" and position in the program sequence.

[0042] Figure 4 depicts an embodiment of the present invention as it would be implemented to assist a run-time execution **3204** executer **2300** of a program listing **2101**. Here, the compiler/interpreter **2303** receives parsed instructions **2202a** and parsed special comments **2202b** which are interpretively executed by the interpretive run time executer **2304**.

[0043] Figure 5 depicts an embodiment of the present invention wherein an example program (program listing) **2101** includes both a basic function "A" **2405** and an optional function "B" **2406**. Executer "A" **2403** is an early version of an executer and doesn't support the optional functions incorporated in the comment fields. Therefore Executer "A" **2403** interprets the

architected statements **2405** in the program listing **2101** and ignores all comment fields **2406**, thereby performing basic function "A" **2405** only. Executer "B" **2404** represents a newer version of the executer which is able to interpret predefined special comments **2406** to perform an optional function "B" **2406**. Therefore Executer "B" **2404** performs not only the basic function **2405** (as does executer "A" **2403**) but also the optional function "B" *2406*. The same example program **2101** runs correctly on both executers but Executer "B" **2404** is able to perform additional function **2406** because of the special comment fields **2406** of the present invention.

Special Comment:

[0044] A "Special Comment" is defined that will preferably, consistently differentiate special comments from ordinary comments in a program listing.

[0045] Special comments can be incorporated in program listings by using techniques other than using program listing comments and be consistent with the teaching of the present invention. An example means to avoid special commenting would be to create a separate listing for the special comments that are with the program listing for editing purposes only. This separate listing optionally cooperates with the program listing and includes special comment function as well as locating means to direct the special comment function to the appropriate program listing function.

[0046] Preferably, special comments will utilize program architecture dependent commenting techniques. An example convention would be to start the comment with "$$" leftmost oriented in the comment field. It should be obvious that other mechanisms could be used to identify the special comment including, but not limited to the use of different special characters that are not ordinarily used in comment field such as "~~"; "###"; "^~^". The options are virtually limitless. Different position, length, number of and content of the characters used to identify the special comment can be chosen than are described herein but would be consistent with the teaching of the present invention. In order to teach the invention, special comments are indicated hereafter

-13-

by leading and trailing explanation marks "!" Within the conventional comment delimiters. Thus special comments take the format of :

/*!xxxxxxxxxxxxxxxx!*/

[0047] In one embodiment, the special comment function is provided entirely by the special comment:

/*!x:type!*/

var x=expression_b;

[0048] In another embodiment, the special comment function is provided by positioning the special comment next to a code statement as:

var x = expression_b /*!type!*/

Where x is implicitly the expression that is assigned "type".

[0049] In another embodiment, the special comment specifies another expression as the type defining expression, thus var x is a type specified in "expression_b" as follows:

var x = expression_c /*!expression_b!*/

[0050] The special comment could provide a level of indirection to the type and be encoded in the comment in a form unrecognizable to the programmer. These and other techniques known in the art are consistent with the teaching of the present invention as long as the comment field participates in the assignment.

Expression Parser:

[0051]   In Figure 6, the expression parser is capable of examining the tokens preceding the cursor **1802** to determine the type of the object (if any) being specified at the cursor **1802**. This is generally done in a language-specific fashion. The expression parser uses information provided by the type manager for making type deductions. Figure 6 shows the effect of the expression parser parsing an expression **1701** using the type defining special comment **1705** to provide the hint window **1801**.

Comment Parser:

[0052]   The Comment Parser is capable of recognizing specific comment formats (such as /*!variablename:type!*/) and keeping a record of the types assigned to variable names. The Comment Parser detects the special comment format (/*!   !*/) and parses the expression within the special comment (variablename:type). This information is used by the expression parser, for example, when deducing the types of expressions involving programmer defined variable names. Additionally, the comment parser can be sensitive to block scoping. That is, comments declaring types of variables that are local to specific blocks of code may only have validity when editing code within those respective blocks. Blocks of code may be identified by beginning and ending special characters such as { } , [ ] or any means known in the art.

Comment Auto-inserter:

[0053]   When the programmer enters assignment statements, i.e., statements that declare and initialize a variable, the editor can use the expression parser to statically determine the type (if any) of the expression being assigned to the variable and generate a special comment to inform the editor in the future of the type assigned to that variable.

[0054]   Figure 7 is an example computer screen **1700** depicting a program listing beginning at **1702** where a type has not been defined for var x **1701**. Figure 8 shows the results of autoinsertion of the special comment defining the var x type as "IHTMLWindoww2" by inserting a type defining special comment **1751** of:

-15-

/!x:IHTMLWindow2!/

Type manager:

[0055]   The type manager is responsible for holding information about the types of objects supplied by the host. Type information includes a specification of the methods, properties, constants, and events associated with a type, as commonly defined by the object oriented programming paradigm.

AutoInsertion:

[0056]   The autoinsertion function is portrayed in the flow diagram in Figure 9. The comment parser is started **1901.** A record is created for the parser to use to understand how to parse the code **1902.** The record contains parameters and conventions such as global variables and comment declarations. Subject source code is parsed by the editor **1912** and resultant tokens are provided to the parser **1903**. As tokens are identified, they are analyzed to determine if they are comments of the special form **1905** according to a comment coding convention that is implementation dependent. If the token identifies a special command, the function in the special command is recorded **1910** in the initial record **1902.** If the code is divided into blocks, a determination is made whether a new block is being entered or exited **1906.** If this is a new block, a child record is created for future special function recording **1910.** All subsequent special form tokens record to this child record until an end of block is detected **1906.** When an end of block is detected, the child record is closed and subsequent special form tokens record to the initial record **1902.** Thus blocks of code are provided with separate special function records **1907,** separate from the global record **1902.**

[0057]   The following example code listing "DHTMLSourceParser.cpp" shows an example implementation of the expression parser of the present invention.

// DHTMLSourceParser.cpp

```
#include "Stdafx.h"
#include "DHTMLSourceParser.h"
#include "SourceEditorView.h"
#include "DHTMLColorCoder.h"
#include "StringBuffer.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
// CJavaScriptParserItem

CJavaScriptParserItem* CJavaScriptParserItem::FindItem(int row, int index)
{
        if (((((row == m_iStartRow) && (index >= m_iStartIndex)) || (row > m_iStartRow)) &&
                (((((row == m_iEndRow) && (index < m_iEndIndex)) || (row < m_iEndRow)))))
        {
                CJavaScriptParserItem* pitem = FindItem(m_aChildren, row, index);
                return pitem ? pitem : this;
        }
        else
                return NULL;
}

CJavaScriptParserItem* CJavaScriptParserItem::FindItem(CJavaScriptParserItemArray& raItems, int row, int index)
{
        // TODO: use binary search
        CJavaScriptParserItem* pitem = NULL;
        for (int i = 0; i < raItems.GetSize(); i++)
        {
                pitem = raItems[i];
                CJavaScriptParserItem* pitem2 = pitem->FindItem(row, index);
                if (pitem2)
                        return pitem2;

                // Have we passed it yet?
                if ((row < pitem->m_iEndRow) || ((row == pitem->m_iEndRow) && (index <=
pitem->m_iEndIndex)))
                        return pitem;
        }

        return pitem;
}

CJavaScriptParserItem* CJavaScriptParserItem::GetFunctionBlockContainer()
{
        CJavaScriptParserItem* pitem = this;
        while (pitem->m_pParent && !dynamic_cast<CJavaScriptFunctionBlock*>(pitem))
                pitem = pitem->m_pParent;
```

```
            return pitem;
    }

    void CJavaScriptParserItem::FindFunctionObjectDeclarations(CMapStringToJavaScriptFunctionBlock&
    rmapFunctions,
            CMapStringToJavaScriptObjectDeclaration& rmapObjects)
    {
            for (int i = 0; i < m_aChildren.GetSize(); i++)
            {
                    CJavaScriptParserItem* pitem = m_aChildren[i];
                    CJavaScriptFunctionBlock* pfunc = dynamic_cast<CJavaScriptFunctionBlock*>(pitem);
                    CJavaScriptObjectDeclaration* pob = dynamic_cast<CJavaScriptObjectDeclaration*>(pitem);

                    if (pfunc)
                            rmapFunctions.SetAt(pfunc->GetContent(), pfunc);
                    else if (pob)
                            rmapObjects.SetAt(pob->GetContent(), pob);
                    else
                            pitem->FindFunctionObjectDeclarations(rmapFunctions, rmapObjects);
            }
    }


    ///////////////////////////////////////////////////////////////
    // CDHTMLSourceParser

    CDHTMLSourceParser::CDHTMLSourceParser(CSourceEditorView* pView) :
            CSourceParser(pView), m_pParseTree(NULL)
    {
            CreateThread();
    }


    CDHTMLSourceParser::~CDHTMLSourceParser()
    {
    }

    void CDHTMLSourceParser::WaitForEvent()
    {
            // Wait for start or destroy event
            HANDLE ah[3];
            ah[0] = m_hEventStart;
            ah[1] = m_hEventDestroy;
            ah[2] = m_hEventStop;

            ::WaitForMultipleObjects(3, ah, FALSE, INFINITE);
    }

    int CDHTMLSourceParser::Run()
    {
            // Acknowledge startup
            ::SetEvent(m_hEventAcknowledge);

            while (true)
            {
                    WaitForEvent();
```

-18-

```
if (m_pParseTree)
        delete m_pParseTree;
EmptyErrorList();
m_pParseTree = NULL;

if (IsDestroyEventSignaled())
        return 0;

if (IsStopEventSignaled())
{
        ::SetEvent(m_hEventFinished);
        ::SetEvent(m_hEventAcknowledge);
        ::ResetEvent(m_hEventStop);
        continue;
}

::ResetEvent(m_hEventFinished);
::SetEvent(m_hEventAcknowledge);

m_pParseTree = new CJavaScriptParserItem;
m_pParseTree->m_iStartRow = m_pParseTree->m_iStartIndex = 0;

// Begin processing
CCell* pcell = m_pView->GetCellManager()->GetCellByIndex(0, 0);
int row = 0, index = 0;
try
{
        ParseCode(pcell, m_pParseTree, row, index);
        m_pParseTree->m_iEndRow = row;
        m_pParseTree->m_iEndIndex = index;
}
catch (int)
{
        // Thrown from GetNextToken indicating termination signal
}

if (IsDestroyEventSignaled())
{
        // Clean up partially completed parse tree
        if (m_pParseTree)
                delete m_pParseTree;
        m_pParseTree = NULL;
        EmptyErrorList();
        return 0;
}

if (pcell == NULL)
{
        // Perform post-processing
}
else
{
        // Clean up partially completed parse tree
        if (m_pParseTree)
                delete m_pParseTree;
```

-19-

```
                    m_pParseTree = NULL;
                    EmptyErrorList();
            }

            ::SetEvent(m_hEventFinished);
            if (IsStopEventSignaled())
            {
                    ::ResetEvent(m_hEventStop);
                    ::SetEvent(m_hEventAcknowledge);
            }
            NotifyFinished();
    }
}

char CDHTMLSourceParser::AdvanceCell(CCell*& rpcell, int& rRow, int& rIndex)
{
        char ch = rpcell->GetChar();
        if ((ch == '\n') || (ch == '\r'))
        {
                ++rRow;
                rIndex = 0;
        }
        else
                ++rIndex;
        rpcell = rpcell->GetNextCell();
        return ch;
}

void CDHTMLSourceParser::GetNextToken(CCell*& rpcell, CString& rstr, int& rType,
                                                        int& rRow, int& rIndex)
{
        CStringBuffer str;

        if (IsStopEventSignaled() || IsDestroyEventSignaled())
                throw (int)0;

        // Take care of special cases
        switch (rpcell->GetColorIndex())
        {
        case DPC_SCRIPT:
                {
                        // What kind of cell is this?
                        char ch = AdvanceCell(rpcell, rRow, rIndex);
                        if (isspace(ch))
                        {
                                // Whitespace
                                rType = DSPC_SCRIPT_WHITESPACE;
                                str += ch;
                                while (rpcell && ((ch = rpcell->GetChar()) != '\n') && isspace(ch))
                                {
                                        str += AdvanceCell(rpcell, rRow, rIndex);
                                }
                        }
                        else if (isalnum(ch) || (ch == '_'))
                        {
```

-20-

```
                    // Identifier
                    rType = DSPC_SCRIPT_IDENTIFIER;
                    str += ch;
                    while (rpcell && (isalnum(ch = rpcell->GetChar()) || (ch == '_')) &&
                            (rpcell->GetColorIndex() == DPC_SCRIPT))
                    {
                            str += AdvanceCell(rpcell, rRow, rIndex);
                    }
            }
            else if (ispunct(ch))
            {
                    // Punctuation
                    rType = DSPC_SCRIPT;
                    str += ch;
            }
            rstr = str;
            return;
    }


case DPC_TAG_ATTRIBUTE_QUOTE:
case DPC_TAG_ATTRIBUTE_DOUBLE_QUOTE:
case DPC_TAG_ATTRIBUTE_QUOTED_CONTENT:
case DPC_TAG_ATTRIBUTE_DOUBLE_QUOTED_CONTENT:
case DPC_TAG_ATTRIBUTE_CLOSE_QUOTE:
        rType = DSPC_HTML_TAG_ATTRIBUTE_VALUE;
        do
        {
                str += AdvanceCell(rpcell, rRow, rIndex);
        }
        while (rpcell &&
                ((rpcell->GetColorIndex() == DPC_TAG_ATTRIBUTE_QUOTE) ||
                (rpcell->GetColorIndex() == DPC_TAG_ATTRIBUTE_DOUBLE_QUOTE) ||
                (rpcell->GetColorIndex() == DPC_TAG_ATTRIBUTE_QUOTED_CONTENT) ||
                (rpcell->GetColorIndex() ==
DPC_TAG_ATTRIBUTE_DOUBLE_QUOTED_CONTENT) ||
                (rpcell->GetColorIndex() == DPC_TAG_ATTRIBUTE_CLOSE_QUOTE)));
        rstr = str;
        return;

case DPC_SCRIPT_QUOTE:
case DPC_SCRIPT_QUOTED_CONTENT:
case DPC_SCRIPT_DOUBLE_QUOTE:
case DPC_SCRIPT_DOUBLE_QUOTED_CONTENT:
case DPC_SCRIPT_CLOSE_QUOTE:
        rType = DSPC_SCRIPT_QUOTATION;
        do
        {
                str += AdvanceCell(rpcell, rRow, rIndex);
        }
        while (rpcell &&
                ((rpcell->GetColorIndex() == DPC_SCRIPT_QUOTE) ||
                (rpcell->GetColorIndex() == DPC_SCRIPT_QUOTED_CONTENT) ||
                (rpcell->GetColorIndex() == DPC_SCRIPT_DOUBLE_QUOTE) ||
                (rpcell->GetColorIndex() == DPC_SCRIPT_DOUBLE_QUOTED_CONTENT) ||
                (rpcell->GetColorIndex() == DPC_SCRIPT_CLOSE_QUOTE)));
```

```
                rstr = str;
                return;

case DPC_SCRIPT_COMMENT:
case DPC_SCRIPT_COMMENT_END:
        rType = DSPC_SCRIPT_COMMENT;
        do
        {
                str += AdvanceCell(rpcell, rRow, rIndex);
        }
        while (rpcell &&
                ((rpcell->GetColorIndex() == DPC_SCRIPT_COMMENT) ||
                (rpcell->GetColorIndex() == DPC_SCRIPT_COMMENT_END)));
        rstr = str;
        return;
}

// Do standard cases: token is just a stream of the same kind of cell
int color = rpcell->GetColorIndex();
switch (color)
{
case DPC_NONE:
        rType = DSPC_HTML;
        break;

case DPC_TAG_NAME:
        rType = DSPC_HTML_TAG_NAME;
        break;

case DPC_TAG_START:
        rType = DSPC_HTML_TAG_START;
        break;

case DPC_TAG_END:
        rType = DSPC_HTML_TAG_STOP;
        break;

case DPC_TAG_ATTRIBUTE_NAME:
        rType = DSPC_HTML_TAG_STOP;
        break;

case DPC_TAG_ATTRIBUTE_EQUAL_SIGN:
case DPC_TAG_SPACE:
        rType = DSPC_HTML_TAG;
        break;

case DPC_COMMENT:
        rType = DSPC_HTML_COMMENT;
        break;

case DPC_SCRIPT_KEYWORD:
        rType = DSPC_SCRIPT_KEYWORD;
        break;

case DPC_SCRIPT_SINGLE_LINE_COMMENT:
```

-22-

```
                rType = DSPC_SCRIPT_COMMENT;
                break;

        default:
                rType = DSPC_UNKNOWN;
        }

        do
        {
                str += AdvanceCell(rpcell, rRow, rIndex);
        }
        while (rpcell && (rpcell->GetColorIndex() == color));

        rstr = str;
}

CJavaScriptParserItem* CDHTMLSourceParser::GetParseTree()
{
        return IsParseComplete() ? m_pParseTree : NULL;
}


void CDHTMLSourceParser::HandleError(CJavaScriptParserItem* pitem, CString strError)
{
        HandleError(strError, pitem->m_iStartRow, pitem->m_iStartIndex, pitem->m_iEndIndex -
pitem->m_iStartIndex);
}

void CDHTMLSourceParser::HandleError(int nBack, int row, int index, CString strError)
{
        HandleError(strError, row, index - nBack, nBack);
}

void CDHTMLSourceParser::ParseFunction(CCell*& rpcell, CJavaScriptParserItem* pitemParent, int& row, int&
index)
{
        // Syntax: function functionname ( parameter, ... ) { body }
        int iStartRow = row;
        int iStartIndex = index;

        // Loop through whitespace and comments
        CString strToken;
        int nType;
        while (rpcell)
        {
                GetNextToken(rpcell, strToken, nType, row, index);

                if (nType == DSPC_SCRIPT_IDENTIFIER)
                        break;

                if ((nType == DSPC_SCRIPT_KEYWORD) || (nType == DSPC_SCRIPT_QUOTATION) ||
                        (nType == DSPC_SCRIPT))
                {
                        HandleError(strToken.GetLength(), row, index, "Invalid function name.");
                        return;
                }
```

-23-

```
        }

        if (rpcell == NULL)
        {
                HandleError("Expected function name.");
                return;
        }

        // Parse function name
        CString strName = strToken;
        bool bFoundParen = false;
        while (rpcell)
        {
                GetNextToken(rpcell, strToken, nType, row, index);

                if (nType == DSPC_SCRIPT_IDENTIFIER)
                        strName += strToken;
                else if ((nType == DSPC_SCRIPT_KEYWORD) || (nType == DSPC_SCRIPT_QUOTATION))
                {
                        HandleError(strToken.GetLength(), row, index, "Invalid function name.");
                        return;
                }
                else if (nType == DSPC_SCRIPT)
                {
                        if (strToken == ".")
                        {
                                if (strName.IsEmpty())
                                {
                                        HandleError(1, row, index, "Function names cannot start with a
period.");

                                        return;
                                }
                                else
                                        strName += ".";
                        }
                        else if (strToken == "(")
                        {
                                bFoundParen = true;
                                break;
                        }
                        else
                        {
                                HandleError(strToken.GetLength(), row, index, "Invalid character in function
name.");

                                return;
                        }
                }
                else if (nType == DSPC_SCRIPT_WHITESPACE)
                        break;
        }

        if (!rpcell)
        {
                HandleError("Expected (.");
                return;
```

-24-

```
                }

        // Parse parameters
        if (!bFoundParen)
        {
                while (rpcell)
                {
                        GetNextToken(rpcell, strToken, nType, row, index);

                        if ((nType == DSPC_SCRIPT_IDENTIFIER) || (nType ==
DSPC_SCRIPT_KEYWORD) || (nType == DSPC_SCRIPT_QUOTATION))
                        {
                                HandleError(strToken.GetLength(), row, index, "Invalid identifier following
function name.");
                                return;
                        }

                        if (nType == DSPC_SCRIPT)
                        {
                                if (strToken == "(")
                                        break;
                                else
                                {
                                        HandleError(strToken.GetLength(), row, index, "Invalid character
following function name.");
                                        return;
                                }
                        }
                }
        }

        CStringArray aParameters;
        if (!rpcell)
        {
                HandleError("Expected {.");
                return;
        }

        bool bLastComma = false;
        bool bFirst = true;
        while (rpcell)
        {
                GetNextToken(rpcell, strToken, nType, row, index);

                if ((nType == DSPC_SCRIPT_KEYWORD) || (nType == DSPC_SCRIPT_QUOTATION))
                {
                        HandleError(strToken.GetLength(), row, index, "Inappropriate function declaration.");
                        return;
                }

                if (nType == DSPC_SCRIPT_IDENTIFIER)
                {
                        if (bFirst || bLastComma)
                        {
                                // Add parameter
```

-25-

```
                                aParameters.Add(strToken);
                        }

                        bFirst = false;
                        bLastComma = false;
                }
                else if (nType == DSPC_SCRIPT)
                {
                        if (strToken == ")")
                        {
                                if (bLastComma)
                                        HandleError(strToken.GetLength(), row, index, "Extra comma.");

                                break;
                        }
                        else if (strToken == ",")
                        {
                                if (bLastComma)
                                        HandleError(strToken.GetLength(), row, index, "Extra comma.");

                                bLastComma = true;
                        }
                        else
                        {
                                HandleError(strToken.GetLength(), row, index, "Invalid character in function
declaration.");

                                return;
                        }
                }
        }

        // Find opening brace
        while (rpcell)
        {
                GetNextToken(rpcell, strToken, nType, row, index);

                if ((nType == DSPC_SCRIPT_IDENTIFIER) || (nType == DSPC_SCRIPT_KEYWORD) ||
(nType == DSPC_SCRIPT_QUOTATION))
                {
                        HandleError(strToken.GetLength(), row, index, "Invalid identifier following function
name.");

                        return;
                }

                if (nType == DSPC_SCRIPT)
                {
                        if (strToken == "{")
                                break;
                        else
                        {
                                HandleError(strToken.GetLength(), row, index, "Invalid character following
function name.");

                                return;
                        }
                }
```

-26-

```
        }

        CJavaScriptFunctionBlock* pfunc = new CJavaScriptFunctionBlock;
        pfunc->m_strContent = strName;
        pfunc->m_iStartRow = iStartRow;
        pfunc->m_iStartIndex = iStartIndex;
        for (int i = 0; i < aParameters.GetSize(); i++)
                pfunc->m_aParameters.Add(aParameters[i]);

        ParseCode(rpcell, pfunc, row, index, false, true);

        pfunc->m_iEndIndex = index;
        pfunc->m_iEndRow = row;
        pitemParent->AddItem(pfunc);
}

void CDHTMLSourceParser::ParseComment(CCell*& rpcell, CString strToken, CJavaScriptParserItem*
pitemParent, int& row, int& index)
{
        CString strHint = strToken;
        if ((strHint.GetLength() <= 4) || (strToken.Left(3) != "//!"))
                return;

        // Format: //!varname:type!
        int i = strToken.Find(':');
        if (i == -1)
                return;

        CString strName = strToken.Mid(3, i - 3);
        CString strType = strToken.Mid(i + 1);
        if ((i = strType.Find('!')) == -1)
                return;
        strType = strType.Left(i);

        CJavaScriptObjectDeclaration* pod = new CJavaScriptObjectDeclaration(strName, strType, row, index,
strToken.GetLength());
        pitemParent->AddItem(pod);
}

void CDHTMLSourceParser::ParseFunctionCall(CCell*& rpcell, CJavaScriptExpression* pexp, int& row, int&
index)
{
        CJavaScriptFunctionCall* pfc = new CJavaScriptFunctionCall;
        pfc->m_iStartIndex = index - 1;
        pfc->m_iStartRow = row;

        CJavaScriptToken* ptok = new CJavaScriptToken("(", row, index);
        pfc->AddItem(ptok);

        while (rpcell)
        {
                CJavaScriptFunctionCallParameter* pparam = new CJavaScriptFunctionCallParameter;
                pparam->m_iStartIndex = index;
                pparam->m_iStartRow = row;
                CString strOut = ParseCode(rpcell, pparam, row, index, true, false, true);
```

```
                pparam->m_iEndRow = row;
                pparam->m_iEndIndex = index;
                pfc->AddItem(pparam);

                if (strOut == ")")
                {
                        // End of function call
                        pfc->m_iEndIndex = index;
                        pfc->m_iEndRow = row;
                        pexp->AddItem(pfc);
                        return;
                }
        }

        // User hasn't typed ) in yet
        pfc->m_iEndIndex = index;
        pfc->m_iEndRow = row;
        pexp->AddItem(pfc);
        return;
}

CString CDHTMLSourceParser::ParseCode(CCell*& rpcell, CJavaScriptParserItem* pitemParent, int& row,
                                                        int& index, bool bParenthesis, bool
bBrace, bool bComma)
{
        CJavaScriptExpression* pLastExpression = NULL;
        bool bLastDot = false;
        while (rpcell)
        {
                // Get the next token
                CString strToken;
                int nType;
                GetNextToken(rpcell, strToken, nType, row, index);

                switch (nType)
                {
                case DSPC_SCRIPT_KEYWORD:
                        if (strToken == "function")
                        {
                                ParseFunction(rpcell, pitemParent, row, index);
                                pLastExpression = NULL;
                        }
                        else
                        {

                        CJavaScriptExpression* pexp = new CJavaScriptExpression(strToken, row,
index);

                        if (bLastDot && pLastExpression)
                        {
                                // Chain this identifier to previous indentifier
                                pexp->m_pPrevious = pLastExpression;
                        }

                        pitemParent->AddItem(pexp);

                        pLastExpression = pexp;
```

```
        }
        bLastDot = false;
        break;

case DSPC_SCRIPT_COMMENT:
        ParseComment(rpcell, strToken, pitemParent, row, index);
        break;

case DSPC_SCRIPT_WHITESPACE:
        break;

case DSPC_SCRIPT:
        if (strToken != "(")
        {
                CJavaScriptToken* pexp = new CJavaScriptToken(strToken, row, index);
                pitemParent->AddItem(pexp);
        }

        if (strToken == ".")
        {
                if (pLastExpression)
                {
                        if (bLastDot)
                                HandleError(pLastExpression, "Syntax error: two consecutive
periods.");
                }
                bLastDot = true;
        }
        else if (strToken == "(")
        {
                if (pLastExpression)
                {
                        ParseFunctionCall(rpcell, pLastExpression, row, index);
                        pLastExpression->m_iEndRow = row;
                        pLastExpression->m_iEndIndex = index;
                }
                else
                {
                        ParseCode(rpcell, pitemParent, row, index, true);

                        CJavaScriptToken* pexp = new CJavaScriptToken(strToken, row,
index);
                        pitemParent->AddItem(pexp);
                }
                pLastExpression = NULL;
                bLastDot = false;
        }
        else if (strToken == "{")
        {
                ParseCode(rpcell, pitemParent, row, index, false, true);
                pLastExpression = NULL;
                bLastDot = false;
        }
        else if (strToken == ")")
        {
```

```
                                if (!bParenthesis)
                                        HandleError(strToken.GetLength(), row, index, "Extra ).");
                                else
                                        return strToken;
                                pLastExpression = NULL;
                                bLastDot = false;
                        }
                        else if (strToken == "}")
                        {
                                if (bBrace)
                                        return strToken;
                                else
                                        HandleError(strToken.GetLength(), row, index, "Extra }.");
                                pLastExpression = NULL;
                                bLastDot = false;
                        }
                        else if ((strToken == ",") && bComma)
                                return strToken;
                        else
                        {
                                pLastExpression = NULL;
                                bLastDot = false;
                                break;
                        }
                        break;

                case DSPC_SCRIPT_IDENTIFIER:
                        {
                                CJavaScriptExpression* pexp = new CJavaScriptExpression(strToken, row,
index);

                                if (bLastDot && pLastExpression)
                                {
                                        // Chain this identifier to previous indentifier
                                        pexp->m_pPrevious = pLastExpression;
                                }

                                pitemParent->AddItem(pexp);

                                pLastExpression = pexp;
                                bLastDot = false;
                                break;
                        }

                case DSPC_SCRIPT_QUOTATION:
                        {
                                CJavaScriptExpression* pexp = new CJavaScriptExpression(strToken, row,
index);

                                pitemParent->AddItem(pexp);

                                pLastExpression = pexp;
                                bLastDot = false;
                                break;
                        }

                default:
```

```
                            if (bParenthesis)
                            {
                                    HandleError(strToken.GetLength(), row, index, "Unexpected end of script.");
                                    return ")";
                            }
                            if (bComma)
                            {
                                    HandleError(strToken.GetLength(), row, index, "Unexpected end of script.");
                                    return ",";
                            }
                            else if (bBrace)
                            {
                                    HandleError(strToken.GetLength(), row, index, "Unclosed brace.");
                                    return "}";
                            }
                    }
            }

            return "";
}

void CDHTMLSourceParser::HandleError(CString strError, int row, int index, int len)
{
            CJavaScriptErrorList* plist;
            if (!m_mapErrors.Lookup(row, plist))
            {
                    plist = new CJavaScriptErrorList;
                    m_mapErrors.SetAt(row, plist);
            }

            CJavaScriptError* perr = new CJavaScriptError;
            perr->m_iStartIndex = index;
            perr->m_iStartRow = row;
            perr->m_iEndIndex = index + len;
            perr->m_strError = strError;
            plist->AddTail(perr);
}

void CDHTMLSourceParser::EmptyErrorList()
{
            POSITION pos = m_mapErrors.GetStartPosition();
            while (pos)
            {
                    int i;
                    CJavaScriptErrorList* plist;
                    m_mapErrors.GetNextAssoc(pos, i, plist);

                    POSITION pos2 = plist->GetHeadPosition();
                    while (pos2)
                    {
                            delete plist->GetNext(pos2);
                    }

                    delete plist;
            }
```

```
        m_mapErrors.RemoveAll();
}

void CDHTMLSourceParser::GetErrorCells(int row, CMap<int, int, int, int>& raIndicies)
{
        if (!IsParseComplete())
                return;

        CJavaScriptErrorList* plist;
        if (m_mapErrors.Lookup(row, plist))
        {
                POSITION pos = plist->GetHeadPosition();
                while (pos)
                {
                        CJavaScriptError* perr = plist->GetNext(pos);
                        for (int i = perr->m_iStartIndex; i < perr->m_iEndIndex; i++)
                                raIndicies.SetAt(i, 0);
                }
        }
}

CString CDHTMLSourceParser::GetError(int row, int index)
{
        if (!IsParseComplete())
                return "";

        CJavaScriptErrorList* plist;
        if (m_mapErrors.Lookup(row, plist))
        {
                POSITION pos = plist->GetHeadPosition();
                while (pos)
                {
                        CJavaScriptError* perr = plist->GetNext(pos);
                        if ((index >= perr->m_iStartIndex) && (index < perr->m_iEndIndex))
                                return perr->m_strError;
                }
        }

        return "";
}
```

Type assistant:

[0058]   The type assistant function is portrayed in Figure 10 in a flow diagram. User input typically from a keypad on a personal computer **2002** is supplied to the autoinsertion compiler, a character at a time **2003.** At an predetermined event (such as the user hi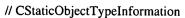tting the "Enter" key) **2004** the autoinsertion compiler will determine the appropriate record for information about the

-32-

code block containing the present location of the cursor **2005.** If the token preceding the Enter operation contained an assignment statement **2006**, then information in the records of the program listing attributes, predefined information such as built-in host object types and other parameters obvious to one skilled in the art, are used to determine if the type of the assigned expression can be identified **2007**. If it can be identified, it is entered into the code listing **2008** as a special command according to the present invention.

[0059]   The following example code listing "TypeInformation.cpp" shows an example implementation of autoinsertion of type information using the present invention.

```
// TypeInformation.cpp

#include "stdafx.h"
#include "resource.h"
#include "TypeInformation.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
// CObjectTypeInformation

CObjectTypeInformation::CObjectTypeInformation()
{
        m_hiconMethod = (HICON)::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(IDI_METHOD),
                IMAGE_ICON, 16, 16, LR_DEFAULTCOLOR);
        m_hiconProperty = (HICON)::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(IDI_PROPERTY),
                IMAGE_ICON, 16, 16, LR_DEFAULTCOLOR);
        m_hiconConstant = (HICON)::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(IDI_CONSTANT),
                IMAGE_ICON, 16, 16, LR_DEFAULTCOLOR);
        m_hiconEvent = (HICON)::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(IDI_EVENT),
                IMAGE_ICON, 16, 16, LR_DEFAULTCOLOR);
}

CObjectTypeInformation::~CObjectTypeInformation()
{
        if (m_hiconMethod)
                ::DestroyIcon(m_hiconMethod);
```

```
            if (m_hiconProperty)
                    ::DestroyIcon(m_hiconProperty);
            if (m_hiconEvent)
                    ::DestroyIcon(m_hiconEvent);
            if (m_hiconConstant)
                    ::DestroyIcon(m_hiconConstant);
}

HICON CObjectTypeInformation::GetItemIcon(int i)
{
        switch (GetMemberType(i))
        {
        case mtFunction:
                return m_hiconMethod;

        case mtEvent:
                return m_hiconEvent;

        case mtConstant:
                return m_hiconConstant;

        default:
                return m_hiconProperty;
        }
}

IMPLEMENT_SERIAL(CMemberInfo, CObject, 0);

void CMemberInfo::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                ar << m_strDescription << m_strName << m_strTypeName << (DWORD)m_type;
        }
        else
        {
                DWORD dw;
                ar >> m_strDescription >> m_strName >> m_strTypeName >> dw;
                m_type = (MemberType)dw;
        }

        m_aParams.Serialize(ar);
}

////////////////////////////////////////////////////////////////////
// CTypeInformation

IMPLEMENT_DYNAMIC(CTypeInformation, CObject);

////////////////////////////////////////////////////////////////////
// CObjectTypeInformation

IMPLEMENT_DYNAMIC(CObjectTypeInformation, CTypeInformation);

////////////////////////////////////////////////////////////////////
```

```
// CStaticObjectTypeInformation

IMPLEMENT_SERIAL(CStaticObjectTypeInformation, CObjectTypeInformation, VERSIONABLE_SCHEMA |
1);

void CStaticObjectTypeInformation::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                ar << m_strName << m_strLibraryName;
        }
        else
        {
                ar >> m_strName >> m_strLibraryName;
        }

        m_aMembers.Serialize(ar);
}

MemberType CStaticObjectTypeInformation::GetMemberType(int i)
{
        return m_aMembers[i]->m_type;
}

CString CStaticObjectTypeInformation::GetName()
{
        return m_strName;
}

CString CStaticObjectTypeInformation::GetMemberTypeName(int i)
{
        return m_aMembers[i]->m_strTypeName;
}

void CStaticObjectTypeInformation::GetMemberParameters(int i, CStringArray& raParams)
{
        raParams.InsertAt(0, &m_aMembers[i]->m_aParams);
}

CString CStaticObjectTypeInformation::GetMemberDescription(int i)
{
        CString str = m_aMembers[i]->m_strDescription;
        switch (m_aMembers[i]->m_security)
        {
        case CMemberInfo::msLow:
                str = "[Low] " + str;
                break;

        case CMemberInfo::msModerate:
                str = "[Moderate] " + str;
                break;

        case CMemberInfo::msSubstantial:
                str = "[Substantial] " + str;
                break;
```

-35-

```
        }

        return str;
}

CString CStaticObjectTypeInformation::GetMemberName(int i)
{
        return m_aMembers[i]->m_strName;
}

int CStaticObjectTypeInformation::FindMember(CString strName)
{
        for (int i = 0; i < m_aMembers.GetSize(); i++)
        {
                if (m_aMembers[i]->m_strName == strName)
                        return i;
        }
        return -1;
}

CStaticObjectTypeInformation::~CStaticObjectTypeInformation()
{
        for (int i = 0; i < m_aMembers.GetSize(); i++)
                delete m_aMembers[i];
        m_aMembers.RemoveAll();
}

void EnumTypeInfoMembers( LPTYPEINFO pITypeInfo, LPTYPEATTR pTypeAttr,
CStaticObjectTypeInformation* pinfo);
//LPCTSTR GetTypeKindName( TYPEKIND typekind );
LPCTSTR GetInvokeKindName( INVOKEKIND invkind );

/////////////////////////////////////////////////////////////////////////////
// CTypeInformationLibrary

IMPLEMENT_SERIAL(CTypeInformationLibrary, CObject, VERSIONABLE_SCHEMA | 1);

CTypeInformationLibrary::CTypeInformationLibrary()
{
}

CTypeInformationLibrary::CTypeInformationLibrary(LPCTSTR lpszTLBPath)
{
        USES_CONVERSION;

        LPTYPELIB pITypeLib;

        HRESULT hr = LoadTypeLib( T2OLE(lpszTLBPath), &pITypeLib );
        if ( S_OK != hr )
        {
//              _tprintf( _T("LoadTypeLib failed on file %s\n"), (LPCTSTR)strFilename );
                AfxThrowMemoryException();
        }

        bool bRet = Init( pITypeLib );
```

-36-

```
        pITypeLib->Release();
        if (!bRet)
                AfxThrowMemoryException();
}

CTypeInformationLibrary::CTypeInformationLibrary(LPTYPELIB pITypeLib)
{
        if (!Init(pITypeLib))
                AfxThrowMemoryException();
}

CTypeInformationLibrary::~CTypeInformationLibrary()
{
        POSITION pos = m_mapTypes.GetStartPosition();
        while (pos != NULL)
        {
                CString strType;
                CTypeInformation* pinfo;
                m_mapTypes.GetNextAssoc(pos, strType, pinfo);
                delete pinfo;
        }
        m_mapTypes.RemoveAll();
}

CTypeInformation* CTypeInformationLibrary::GetTypeInformation(CString strTypeName)
{
        CTypeInformation* pob = NULL;
        m_mapTypes.Lookup(strTypeName, pob);
        return pob;
}

void CTypeInformationLibrary::RegisterTypeInformation(CTypeInformation* pinfo)
{
        m_mapTypes.SetAt(pinfo->GetName(), pinfo);
}

void CTypeInformationLibrary::Serialize(CArchive& ar)
{
        m_mapTypes.Serialize(ar);

        if (ar.IsStoring())
                ar << m_strName;
        else
                ar >> m_strName;
}

/*void CTypeInformationRegistry::LoadTypeInformation(CString strFilename)
{
        CFile file(strFilename, CFile::modeRead);
        CArchive ar(&file, CArchive::load);
        CTypeInformationRegistry reg;
        reg.Serialize(ar);

        POSITION pos = reg.m_mapTypes.GetStartPosition();
```

-37-

```
        while (pos != NULL)
        {
                CString strType;
                CTypeInformation* pinfo;
                reg.m_mapTypes.GetNextAssoc(pos, strType, pinfo);
                TRACE("Registered %s\n", (LPCTSTR)strType);
                m_mapTypes.SetAt(strType, pinfo);
        }

        reg.m_mapTypes.RemoveAll();
}
*/

bool CTypeInformationLibrary::Init( LPTYPELIB pITypeLib )
{
        BSTR pszTypeInfoName;
        if (SUCCEEDED(pITypeLib->GetDocumentation(-1, &pszTypeInfoName, 0, 0, 0)))
        {
                m_strName = BSTRToCString(pszTypeInfoName);
                ::SysFreeString(pszTypeInfoName);
        }

        UINT c = pITypeLib->GetTypeInfoCount();

        for (UINT i = 0; i < c; i++)
        {
                LPTYPEINFO pITypeInfo;

                HRESULT hr = pITypeLib->GetTypeInfo(i, &pITypeInfo);

                if (hr == S_OK)
                {
                        ImportTypeInfo(pITypeInfo);
                        pITypeInfo->Release();
                }
                else
                        return false;
        }

        return true;
}

void CTypeInformationLibrary::ImportTypeInfo(LPTYPEINFO pITypeInfo)
{
        HRESULT hr;

        BSTR bstrTypeInfoName, bstrTypeDescription;
        hr = pITypeInfo->GetDocumentation(MEMBERID_NIL, &bstrTypeInfoName, &bstrTypeDescription,
                NULL, NULL);
        if (hr != S_OK)
                return;

        _bstr_t strTypeInfoName(bstrTypeInfoName, FALSE), strTypeDescription(bstrTypeDescription, FALSE);

        TYPEATTR* pTypeAttr;
```

-38-

```
        hr = pITypeInfo->GetTypeAttr(&pTypeAttr);
        if (S_OK != hr)
                return;

        if (pTypeAttr->typekind == TKIND_DISPATCH)
        {
                CStaticObjectTypeInformation* pinfo = new CStaticObjectTypeInformation;
                pinfo->m_strName = (LPCTSTR)strTypeInfoName;
                pinfo->m_strDescription = (LPCTSTR)strTypeDescription;
                pinfo->m_strLibraryName = m_strName;
                EnumTypeInfoMembers( pITypeInfo, pTypeAttr, pinfo );
                RegisterTypeInformation(pinfo);
        }

        pITypeInfo->ReleaseTypeAttr(pTypeAttr);
}

void Add(CMemberInfo* pmember, CStaticObjectTypeInformation* pinfo)
{
        // Eliminate these members:
        // QueryInterface
        // AddRef
        // Release
        // GetTypeInfoCount
        // GetTypeInfo
        // GetIDsOfNames
        // Invoke

        if ((pmember->m_strName == "QueryInterface") ||
                (pmember->m_strName == "AddRef") ||
                (pmember->m_strName == "Release") ||
                (pmember->m_strName == "GetTypeInfoCount") ||
                (pmember->m_strName == "GetTypeInfo") ||
                (pmember->m_strName == "GetIDsOfNames") ||
                (pmember->m_strName == "Invoke") ||
                ((pmember->m_strName.GetLength() > 0) && (pmember->m_strName.GetAt(0) == '_')))
        {
                delete pmember;
                return;
        }


        for (int i = 0; i < pinfo->m_aMembers.GetSize(); i++)
        {
                if (pinfo->m_aMembers[i]->m_strName == pmember->m_strName)
                {
                        delete pmember;
                        return;
                }
        }

        //cout << (LPCTSTR)pmember->m_strTypeName << " " << (LPCTSTR)pmember->m_strName << endl;
        pinfo->m_aMembers.Add(pmember);
}
```

```
//=================================================================
// Given an ITypeInfo instance, retrieve the name.
//=================================================================
void GetTypeInfoName( LPTYPEINFO pITypeInfo, LPTSTR pszName, MEMBERID memid )
{
        USES_CONVERSION;
        BSTR pszTypeInfoName;
        HRESULT hr;

        hr = pITypeInfo->GetDocumentation( memid, &pszTypeInfoName, 0, 0, 0 );

        if ( S_OK != hr )
        {
                lstrcpy( pszName, _T("<unknown>") );
                return;
        }

        // Make a copy so that we can free the BSTR allocated by ::GetDocumentation
        lstrcpy( pszName, OLE2T(pszTypeInfoName) );

        // Free the BSTR allocated by ::GetDocumentation
        SysFreeString( pszTypeInfoName );
}

CString GetTypeName(LPTYPEINFO pinfo, TYPEDESC* pdesc)
{
        while (pdesc->vt == VT_PTR)
        {
                pdesc = pdesc->lptdesc;
        }

        switch (pdesc->vt)
        {
        case VT_USERDEFINED:
                {
                        LPTYPEINFO pinfo2;
                        if (SUCCEEDED(pinfo->GetRefTypeInfo(pdesc->hreftype, &pinfo2)))
                        {
                                char achBuffer[1024];
                                GetTypeInfoName(pinfo2, achBuffer, MEMBERID_NIL);
                                pinfo2->Release();
                                return CString(achBuffer);
                        }
                        return "";
                }

        default:
                return "";
        }
}

void EnumTypeInfoMembers( LPTYPEINFO pITypeInfo, LPTYPEATTR pTypeAttr,
CStaticObjectTypeInformation* pinfo )
{
        USES_CONVERSION;
```

```
CMemberInfo* pmember;
if ( pTypeAttr->cFuncs )
{
        //TRACE("Functions:\n");

        for ( unsigned i = 0; i < pTypeAttr->cFuncs; i++ )
        {
                pmember = new CMemberInfo;
                FUNCDESC * pFuncDesc;

                pITypeInfo->GetFuncDesc( i, &pFuncDesc );

                BSTR pszFuncName;
                BSTR doc;
                pITypeInfo->GetDocumentation(pFuncDesc->memid, &pszFuncName,&doc,0,0);

                pmember->m_strName = OLE2T(pszFuncName);
                pmember->m_strDescription = OLE2T(doc);
                SysFreeString(doc);

                // Look for <sashhelp
                if ((pmember->m_strDescription.GetLength() > 9) &&
                        !pmember->m_strDescription.Left(9).CompareNoCase("<sashhelp"))
                        pmember->ParseSashHelpXML(pmember->m_strDescription);
                else
                        pmember->m_type = (pFuncDesc->invkind == INVOKE_FUNC ? mtFunction :
                                pFuncDesc->cParams > 0 ? mtFunction : mtObject);

                pmember->m_strTypeName = GetTypeName(pITypeInfo,
&pFuncDesc->elemdescFunc.tdesc);

                //TRACE( _T("    %-32ls\n"), pszFuncName );

                // Enumerate parameters
                BSTR* pstrParams = new BSTR[pFuncDesc->cParams + 1];
                UINT u;
                if (SUCCEEDED(pITypeInfo->GetNames(pFuncDesc->memid, pstrParams,
pFuncDesc->cParams + 1, &u)))
                {
                        if (u)
                                ::SysFreeString(pstrParams[0]);
                        for (UINT k = 1; k < u; k++)
                        {
                                CString strType = OLE2T(pstrParams[k]);
                                //TRACE("        %s\n", (LPCTSTR)strType);
                                ::SysFreeString(pstrParams[k]);
                                pmember->m_aParams.Add(strType);
                        }
                }
                delete [] pstrParams;

                pITypeInfo->ReleaseFuncDesc( pFuncDesc );

                SysFreeString( pszFuncName );
```

-41-

```
                            Add(pmember, pinfo);
                    }
            }

            if ( pTypeAttr->cVars )
            {
                    //TRACE("Variables:\n");

                    for ( unsigned i = 0; i < pTypeAttr->cVars; i++ )
                    {
                            pmember = new CMemberInfo;
                            VARDESC * pVarDesc;

                            pITypeInfo->GetVarDesc( i, &pVarDesc );

                            BSTR pszVarName;
                            pITypeInfo->GetDocumentation(pVarDesc->memid, &pszVarName,0,0,0);

                            pmember->m_strName = OLE2T(pszVarName);
                            pmember->m_type = mtObject;
                            pmember->m_strTypeName = GetTypeName(pITypeInfo,
&pVarDesc->elemdescVar.tdesc);
                            //TRACE( _T("    %ls\n"), pszVarName );

                            pITypeInfo->ReleaseVarDesc( pVarDesc );
                            SysFreeString( pszVarName );
                            Add(pmember, pinfo);
                    }
            }

    }

void CMemberInfo::ParseSashHelpXML(LPCTSTR lpszXML)
{
            // Parse documentation string
            try
            {
                    MSXML::IXMLDOMDocumentPtr pdoc(__uuidof(MSXML::DOMDocument));
                    pdoc->loadXML(lpszXML);

                    MSXML::IXMLDOMNamedNodeMapPtr pAttributes = pdoc->documentElement->attributes;

                    // Determine type
                    MSXML::IXMLDOMNodePtr pattrib = pAttributes->getNamedItem("type");
                    if (pattrib.GetInterfacePtr())
                    {
                            CString strType = (LPCTSTR)_bstr_t(pattrib->nodeValue);

                            if (strType == "Method")
                                    m_type = mtFunction;
                            else if (strType == "Event")
                                    m_type = mtEvent;
                            else if (strType == "Constant")
                                    m_type = mtConstant;
                            else if (strType == "Constructor")
```

-42-

```
                              m_type = mtConstructor;
            }

            // Determine security
            pattrib = pAttributes->getNamedItem("security");
            if (pattrib.GetInterfacePtr())
            {
                    CString strSecurity = (LPCTSTR)_bstr_t(pattrib->nodeValue);

                    if (strSecurity == "Low")
                            m_security = CMemberInfo::msLow;
                    else if (strSecurity == "Moderate")
                            m_security = CMemberInfo::msModerate;
                    else if (strSecurity == "Substantial")
                            m_security = CMemberInfo::msSubstantial;
            }

            // Parse event handler prototype
            // of the form:
            // <eventhandler><parameter type=\"LONG\" name=\"cx\"/>
            // <parameter type=\"LONG\" name=\"cy\"/></eventhandler>

            // Retrieve description
            MSXML::IXMLDOMElementPtr pHelpMsg =
pdoc->documentElement->selectSingleNode("helpstring");
                    if (pHelpMsg.GetInterfacePtr())
                            m_strDescription = (LPCTSTR)pHelpMsg->text;

            // Retrieve parameters
            MSXML::IXMLDOMNodeListPtr pParams =
pdoc->documentElement->selectNodes("parameters/parameter");
                    if (pParams.GetInterfacePtr())
                    {
                            long c = pParams->length;
                            for (long i = 0; i < c; i++)
                            {
                                    MSXML::IXMLDOMElementPtr pParam = pParams->item[i];
                                    if (pParam.GetInterfacePtr())
                                    {
                                            MSXML::IXMLDOMNamedNodeMapPtr pAttributes =
pParam->attributes;

                                            // Determine name
                                            MSXML::IXMLDOMNodePtr pattrib =
pAttributes->getNamedItem("name");

                                            if (pattrib.GetInterfacePtr())
                                                    m_aParams.Add((LPCTSTR)pattrib->text);
                                    }
                            }
                    }
            }
            catch (_com_error&)
            {
            }
}
```

-43-

```
CString CTypeInformationLibrary::GetNextType(POSITION& rpos, CTypeInformation** ppinfo)
{
        CString str;
        CTypeInformation* pinfo;
        m_mapTypes.GetNextAssoc(rpos, str, pinfo);

        if (ppinfo)
                *ppinfo = pinfo;
        return str;
}

void CTypeInformationLibrary::DeleteAll()
{
        POSITION pos = GetFirstTypePosition();
        while (pos)
        {
                CTypeInformation* pinfo;
                GetNextType(pos, &pinfo);
                delete pinfo;
        }

        m_mapTypes.RemoveAll();
}

void CTypeInformationLibrary::RenameType(CString strOldName, CString strNewName)
{
        CStaticObjectTypeInformation* pinfo =
dynamic_cast<CStaticObjectTypeInformation*>(GetTypeInformation(strOldName));
        pinfo->m_strName = strNewName;
        m_mapTypes.RemoveKey(strOldName);
        m_mapTypes.SetAt(strNewName, pinfo);
}

//////////////////////////////////////////////////////////////////////
// CTypeInformationRegistry

CTypeInformationRegistry::CTypeInformationRegistry()
{
        m_pChain = NULL;
}

CTypeInformationRegistry::CTypeInformationRegistry(const CTypeInformationRegistry& r)
{
        CopyRegistry(r);
        m_pChain = NULL;
}

CTypeInformationRegistry::CTypeInformationRegistry(CTypeInformationRegistry* pChain)
{
        ChainRegistry(pChain);
}

CTypeInformation* CTypeInformationRegistry::GetTypeInformation(CString strTypeName) const
{
```

-44-

```
        // Search through the libraries
        POSITION pos = GetFirstLibraryPosition();
        while (pos)
        {
                CTypeInformationLibrary* plib = GetNextLibrary(pos);
                CTypeInformation* pinfo = plib->GetTypeInformation(strTypeName);
                if (pinfo)
                        return pinfo;
        }
        return m_pChain ? m_pChain->GetTypeInformation(strTypeName) : NULL;
}


CTypeInformationLibrary* CTypeInformationRegistry::GetLibrary(LPCTSTR lpszLibraryName) const
{
        CTypeInformationLibrary* plib;
        return m_mapLibraries.Lookup(lpszLibraryName, plib) ? plib :
                (m_pChain && (plib = m_pChain->GetLibrary(lpszLibraryName)) ? plib : NULL);
}


void CTypeInformationRegistry::AddLibrary(CTypeInformationLibrary* plib)
{
        m_mapLibraries.SetAt(plib->GetLibraryName(), plib);
}


CTypeInformationLibrary* CTypeInformationRegistry::RemoveLibrary(CString strLibrary)
{
        CTypeInformationLibrary* plib = GetLibrary(strLibrary);
        m_mapLibraries.RemoveKey(strLibrary);
        return plib;
}


CTypeInformationLibrary* CTypeInformationRegistry::AddLibrary(LPCTSTR lpszTLBPath)
{
        try
        {
                CTypeInformationLibrary* plib;
                AddLibrary(plib = new CTypeInformationLibrary(lpszTLBPath));
                return plib;
        }
        catch (CException* p)
        {
                p->Delete();
                return NULL;
        }
}


void CTypeInformationRegistry::CopyRegistry(const CTypeInformationRegistry& r)
{
        POSITION pos = r.GetFirstLibraryPosition();
        while (pos)
                AddLibrary(r.GetNextLibrary(pos));
}


void CTypeInformationRegistry::ChainRegistry(CTypeInformationRegistry* pChain)
{
```

```
        m_pChain = pChain;
}

void CTypeInformationRegistry::DeleteAll()
{
        POSITION pos = GetFirstLibraryPosition();
        while (pos)
                delete GetNextLibrary(pos);

        m_mapLibraries.RemoveAll();
}
```

[0060]   Although preferred embodiments have been depicted and described in detail herein, it will be apparent to those skilled in the relevant art that various modifications, additions, substitutions and the like can be made without departing from the spirit of the invention, and these are therefore considered to be within the scope of the invention as defined in the following claims: